30 | 如何确定生产系统配置？

2021-05-31 高楼

《高楼的性能工程实战课》 课程介绍 >

讲述：高楼

时长 25:15 大小 23.13M

你好，高楼。

在性能“测试”的范畴中，配置生产系统一直都是运维的活，和我们“测试”没啥关系。

但是，我在第一节课里就强调，在我的 RESAR 性能工程理念中，性能工程要考虑到运维阶段。这看似是一个比较小的改变，但实际上延展了性能团队的工作范围，执行起来并不容易，尤其是对于那些运维和性能“测试”团队严重脱节的企业。

我们暂且不说性能“测试”团队能不能给出生产上想要的配置，很多性能“测试”团队，能连当前生产的配置都不知道。面对这样的情况，我认为如果我们还龟缩在“测试”团队中，就必然做不出什么贡献了。

我们想想性能项目的目标，就很容易理解这一点。通常我们在制定目标的时候，会有这样的说法：保证线上系统正常运行。

这个目标看起来应该在性能项目中完成，可是，在当前的性能行业中，又是怎么做的呢？如果你是一个性能“测试”工程师，是不是连生产的样子都没有见过？连数据也没有拿到过？性能参数也没有分析过？更有甚者，可能连机器都没有见过。在这样的情形之下，性能项目也只能找一些系统上明显的软件性能瓶颈而已。

而一个系统整体的容量，绝对不是仅由软件组成的，还有硬件环境、网络、存储、负载均衡、防火墙等等一系列的软硬件。如果性能团队对这些都不了解，那就不能指望他们可以给出什么生产配置。

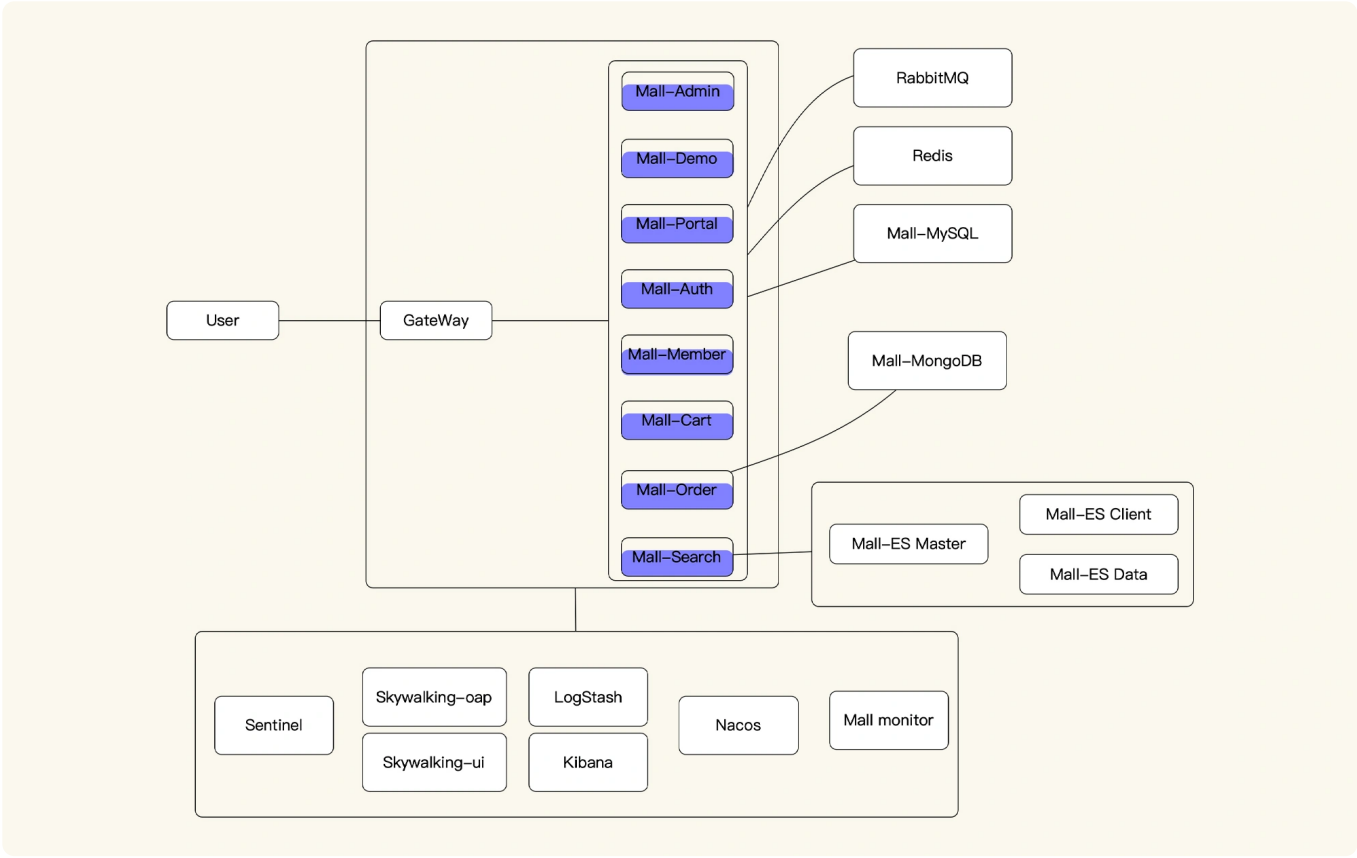
当我们把这个问题后移到生产环境中时，运维团队有经验的人也许可以给出合理的性能参数配置。但是，这些参数配置是否和现在的业务目标匹配呢？可能大部分运维会先上线，然后再调优校准参数。而这样就意味着，系统在上线一开始是不稳定的。

所以，在我看来，**应该由性能团队给出生产环境中的性能参数配置**，这是最为合理的。

预判生产容量

在确定性能参数配置之前，我们要先预判生产的大概容量，不用特别精确，像“在1000TPS 左右”这样的预估就可以了。其实，这就是预估一个系统的容量水位。

就如这张图所示，我们要先大致估计出每个服务在不同的容量之下，会使用到多少的资源。然后尽量让资源均衡使用，减少成本。



经常会有人问这样的问题：怎么评估一个系统的容量？比如说，我们拿到一个 4C8G 的机器配置，在一个我们测试过的系统中，怎么评估这个机器能跑出多少 TPS？

其实，我们可以从最简单的做起：基准测试。

之前，有一个学员问我，一个 8C16G 的机器能跑出多少 TPS？我回答说不知道。因为我不清楚是什么业务，如果是我没有测试过的业务，那我就更没有什么经验数据了。所以，我建议她去做一下基准测试，哪怕是最简单的没有业务逻辑的 CRUD 服务，也能知道跑出多少 TPS。

根据我的经验，在我的一个 2C4G 的机器上，如果只跑最简单的查询接口，并且没有任何业务逻辑，那跑出 1000TPS（一个 T 就是一次接口请求）是没问题的。

那个学员也比较认真，回去就弄了一个简单的服务试了一下，然后告诉我 8C16G 的机器能跑出三、四千的 TPS。这个结果和我的经验结果差不多，因为她的环境是我的四倍，跑出来的 TPS 也能达到我的四倍。

不过，这其中有一个很明显的问题，就是这个实验示例没有业务逻辑。**对于有业务逻辑的业务系统来说，最大容量取决于业务的复杂度。**所以，我在进到一个新项目中时，通常都

会先了解一下历史性能数据，再来判断是否有优化的必要。对于我了解的系统，在知道了硬件和软件架构之后，我心里大概能有一个预期目标。

对于不了解的系统，我们也不难得到最大容量的数据，只要做一下容量场景就可以知道了。

当然，在一个生产系统中有相应的判断能力。笼统地说就是，如果有 1000C 2.5GHz 的 CPU 资源，我们要根据历史经验数据，判断出最大容量能跑多少 TPS；如果是 2000C 2.5GHz 的 CPU 资源，又能跑出多少 TPS。而这些都可以通过容量场景计算出来。

之所以是“笼统地说“，是因为最大容量和很多细节都有关系，比如架构设计的合理性、预留多少生产资源等方方面面。因此，并没有一套所谓标准的配置，可以适配于任何一个系统。

可能有人会问，通过容量场景计算出 TPS 之后，是不是可以再用排队论模型，来计算需要多少服务器资源呢？这个逻辑的确行得通，不过需要先建模，并采样大量的数据来做计算。这个话题很大，我在这里不展开讨论了，但你可以知道有这么一个方向。

而在这节课中，我希望能通过实践让你明白获得合理配置的逻辑。

你还记得这个性能分析决策树吗？

图中这些是在我这个课程的示例系统中使用的各种组件。对应各个组件，我们都应该给出合理的性能配置。

那性能配置主要是指哪些方面呢？我们要分为硬件和软件两大角度来看。

硬件配置

硬件配置其实是很大一块内容，通常，我们都会在测试环境中受到硬件资源的限制。因此，我们会这样来计算大概的容量：

1. 拿到生产环境的硬件配置，以及峰值场景下的资源利用率、TPS、RT 数据。
2. 在测试环境硬件配置下，通过容量场景，得到测试环境中的峰值场景下的资源利用率、TPS、RT 数据。
3. 拿第一步和第二步中得到的数据做对比。

通过这三个步骤，我们就能知道在生产环境中，系统所能支撑的最大 TPS 大概是多少。如果列一个简单的示例表格，那就是这样：

环境	TPS	响应时间	CPU资源	CPU资源使用率
生产环境	10000	0.1s	1000C	30%
测试环境	10000	0.1s	300C	100%

也就是说，如果在生产环境用 1000C 的 30%，同时容量可以达到 10000TPS，平均响应时间可以达到 0.1s，那么在测试环境中，我们至少在 300C 的使用率达到 100% 的时候，容量才能达到 10000TPS、平均响应时间 0.1s。

当然，你可以有一百种理由说我这个逻辑不合理，比如说，最明显的问题就是 CPU 用到 100%，业务系统显然不稳定，并且 TPS 的增加也不可能是线性的；这里没考虑到其他的硬件资源情况等等。

没错，这显然是一个非常粗糙的计算过程，而我在这里也只是为了给你举一个例子。在你真正做计算时，可以把相应的重要资源都列上去。而这个建模过程需要拿大量的样本数据做分析。

我们用一个表格来大概建模，计算一下不同环境的资源产生的 TPS 比对：

环境	TPS	响应时间	CPU资源	CPU资源使用率
生产环境	10000	0.1s	1000C	30%
测试环境	?	0.1s	300C	30%

如果我们测试环境有 300C 资源，使用率也为 30%，要是我还想保证 0.1 秒的平均响应时间，那么 TPS 就应该是 3000。这是最简单的等比方式了。

但是，硬件的不同有很多因素，所以，我们要在一个项目中要建模才可以。而建模要考虑的因素只有从具体的项目中才能拿到，大概有这几点：

1. 硬件、软件配置；
2. 生产环境和测试环境的 TPS、RT 数据；
3. 生产环境和测试环境的资源利用率数据（用性能决策树中的全局计数器）。

因为每个业务系统消耗的资源会有偏向，要么是计算密集型，要么是 IO 密集型，所以，我们在比对计数器的时候，肯定要比对那些消耗得快的计数器。

拿到上面这些数据后，我们再创建上面表格中的等比模型，就可以计算测试环境中的最大容量了。

但是这个数据仍然不够完整，因为我们还要关注软件配置。

软件配置

对于软件配置，也同样需要我们做相应的等比计算。我们扩展一下上面的表格：

环境	TPS	响应时间	CPU资源	CPU资源使用率	Java服务线程数
生产环境	10000	0.1s	1000C	30%	1000
测试环境	10000	0.1s	300C	100%	300

如果我们在测试环境中达到了硬件配置，没达到软件配置，就像下面表格这样，我们该怎么计算测试环境中的 TPS 和资源使用率呢？

环境	TPS	响应时间	CPU资源	CPU资源使用率	Java服务线程数
生产环境	10000	0.1s	1000C	30%	1000
测试环境	?	0.1s	300C	?	100

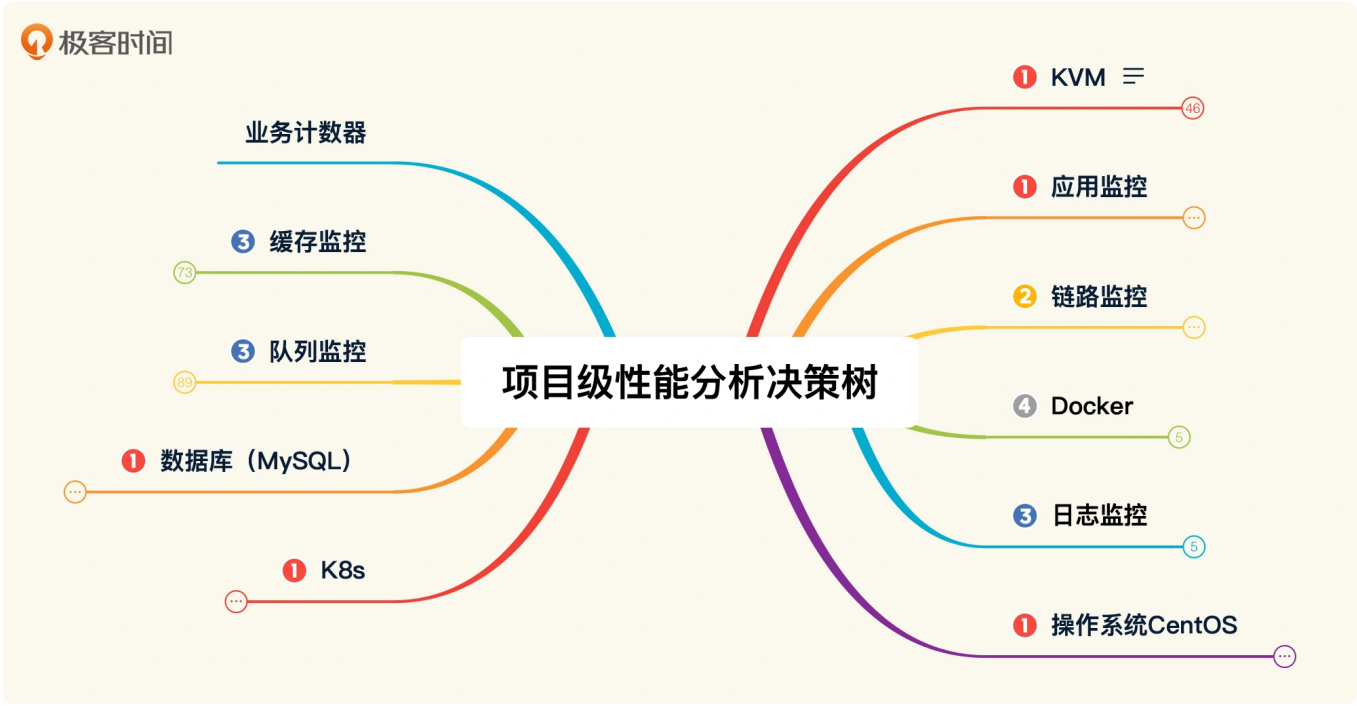
显然，这时候表格中两个问号代表的数据就不一样了。通过计算你就可以知道，测试环境要想达到 1000TPS，而资源使用率也只能用到 1/10（也就是 30C）了。

当然，实际的建模过程不会这么简单，不会只靠这么一两个计数器就能完成。那我们在实际建模过程中，应该把哪些计数器纳入到计算当中呢？这就涉及到性能分析决策树中，所有的性能计数器了。而这些计数器会和相对应的性能配置相关。因此，我们要对应性能分析决策树，我画一个性能配置树出来。

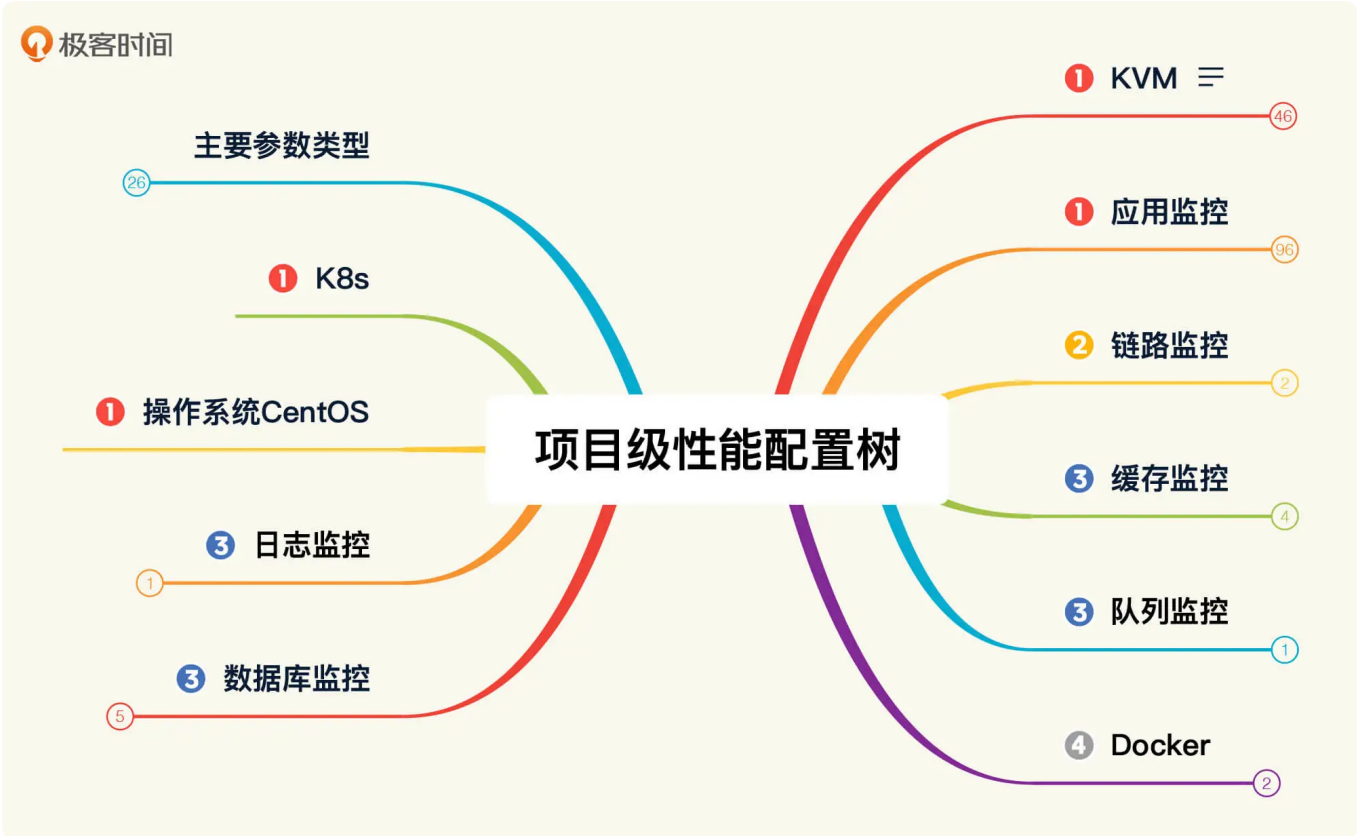
性能配置树

对应前面的性能分析决策树，我们画一个性能配置树。

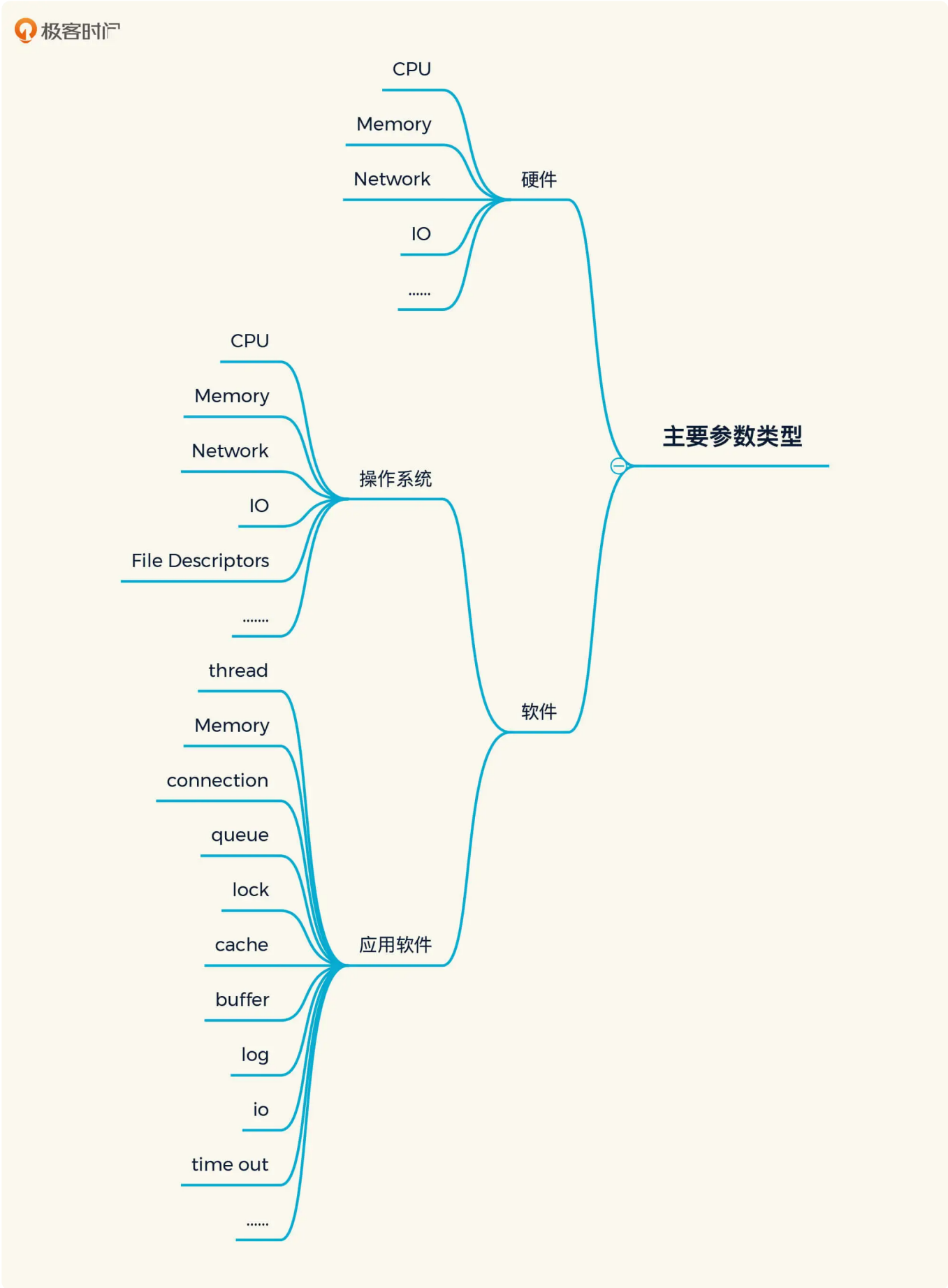
性能分析决策树：



性能配置树：



通过对比，相信你已经发现，我在性能配置树中加了一个“主要参数类型”。把“主要参数类型”展开之后，我们可以看到这样的列表：



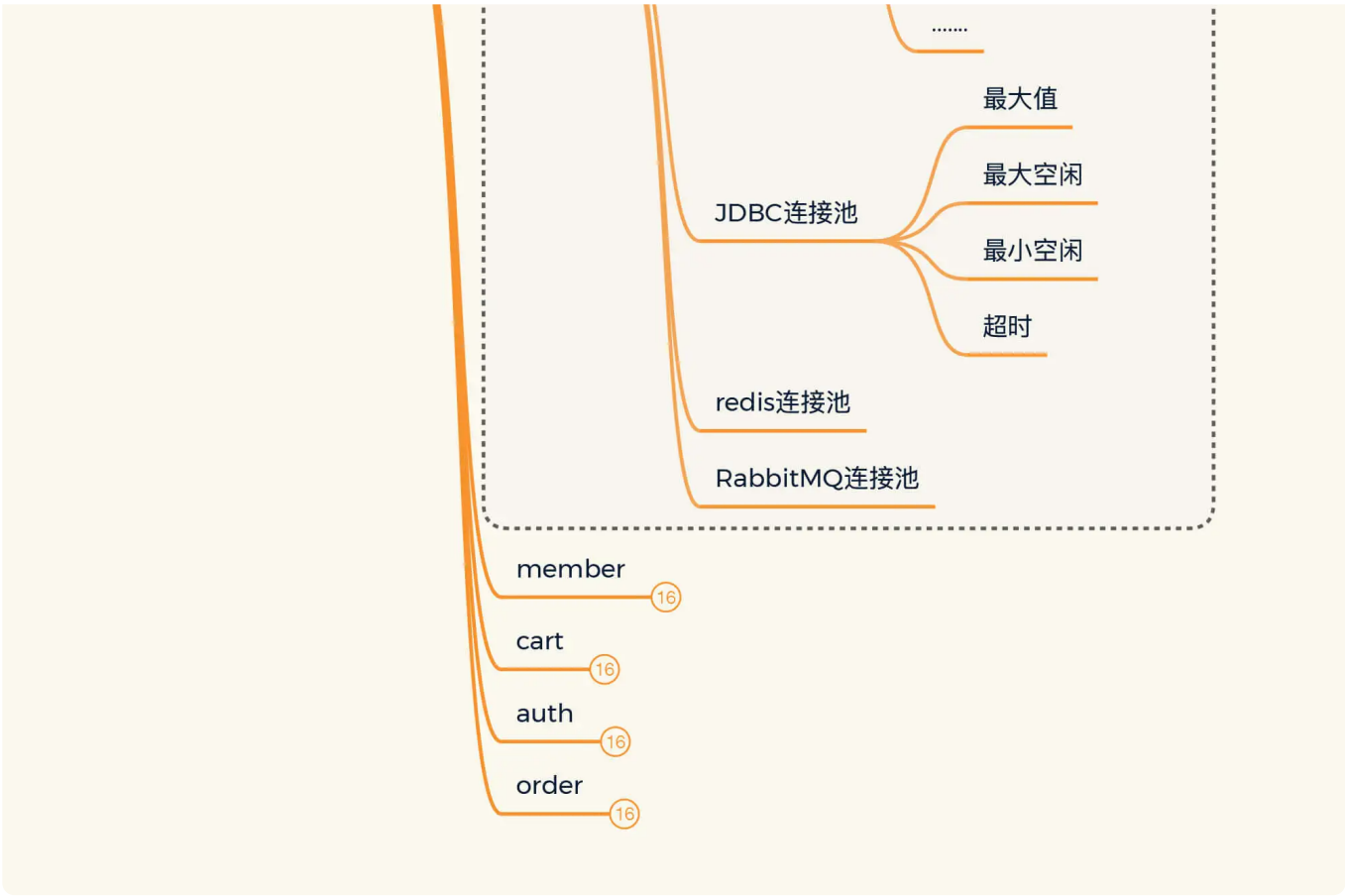
其中，硬件包含的参数和操作系统包含的参数看上去一样，不过，我们实际上要对比的内容并不一样。比如说 CPU，在硬件的层面，我们要对比的是型号、主频、核数 /NUMA 等

信息；而在软件层面，我们要对比的则是 CPU 使用率。其他的性能参数和计数器也有类似区别。

而在应用软件方面，我罗列了最常见的比对参数，也就是说在每一个软件技术组件中，我们都要从这些角度去考虑需要提取的配置。

在这里，我要说明一下，我在性能配置树中描述的是一种通用特征，因此无法对每个组件的配置都那么面面俱到。在具体的技术组件中，需要你灵活更改。就以 MySQL 为例，对于内存，我们通常会考虑 innodb_buffer_pool_size；而对于 java 微服务，我们通常是用 JVM 来表达。

所以，针对性能配置树的每一个技术组件，我们还需要细化，就拿最常见的 Java 微服务应用来说，我们要考虑的范围如下图所示：



由于参数太多，无法在图中完全表达出来，我直接用省略号代替了。对于其他技术组件，我们也要像这样一一列出重要的配置。

在这里，我给你一个常见的各系统性能参数表格，同时，我也把完整的性能配置树也放在一起了，供你参考。点击此处就能下载，密码为 4f6u。

在这个文件中，并非所有的参数都与性能相关，你只需要根据我前面说的类型（比如线程数、超时、队列、连接、缓存等）自己筛选就好了。另外，我根据自己的工作经验，把其中重要的参数都标红了，当然这也只是给你借鉴。在你自己的项目中，你可以按性能配置树中的逻辑罗列自己的参数列表。

讲到这里，我们就要进入下一步了：获得这些参数在生产环境中的具体配置值。

如何获取配置值

获取配置值的方法主要分为两个步骤：

1. 运行场景；

2. 查看相应的计数器。

现在，我们就以 Order 服务为例，看看到底怎么确定相关参数的配置值。

压力场景数据

我们先执行性能项目中的容量场景，判断一下 TPS 大概能达到多少。

在这个场景中，你可以看到，在 30 压力线程时，TPS 大概能达到 800 左右。但是，随着压力的增加，TPS 也能达到 1000，只是响应时间也随之有了明显的递增趋势。

接下来，我们就分析一下这个状态需要什么样的配置。

由于配置太多，而确认配置又是一个非常细致的工作，我们不太可能尽述。不过，我会告诉你确定配置的逻辑是什么。这样，你在自己的项目中，都可以按这个逻辑来确定每个技术组件的相关性能参数。

应用服务的线程数配置

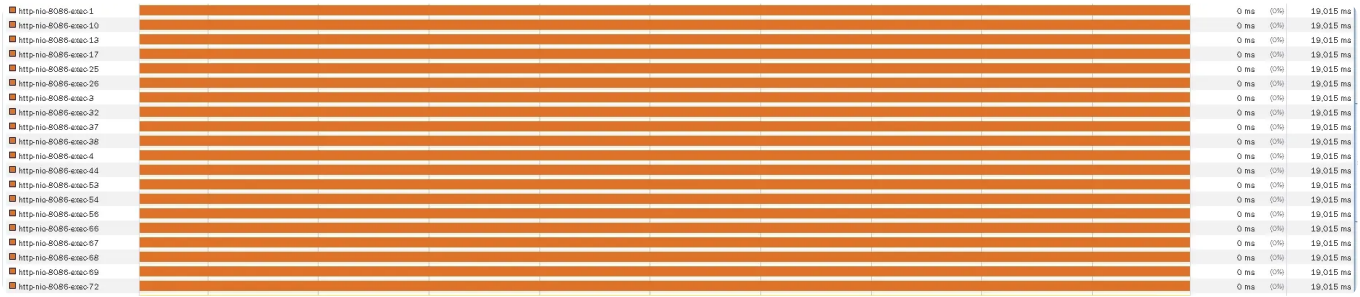
我们先看看 Order 的当前配置是什么样的：

复制代码

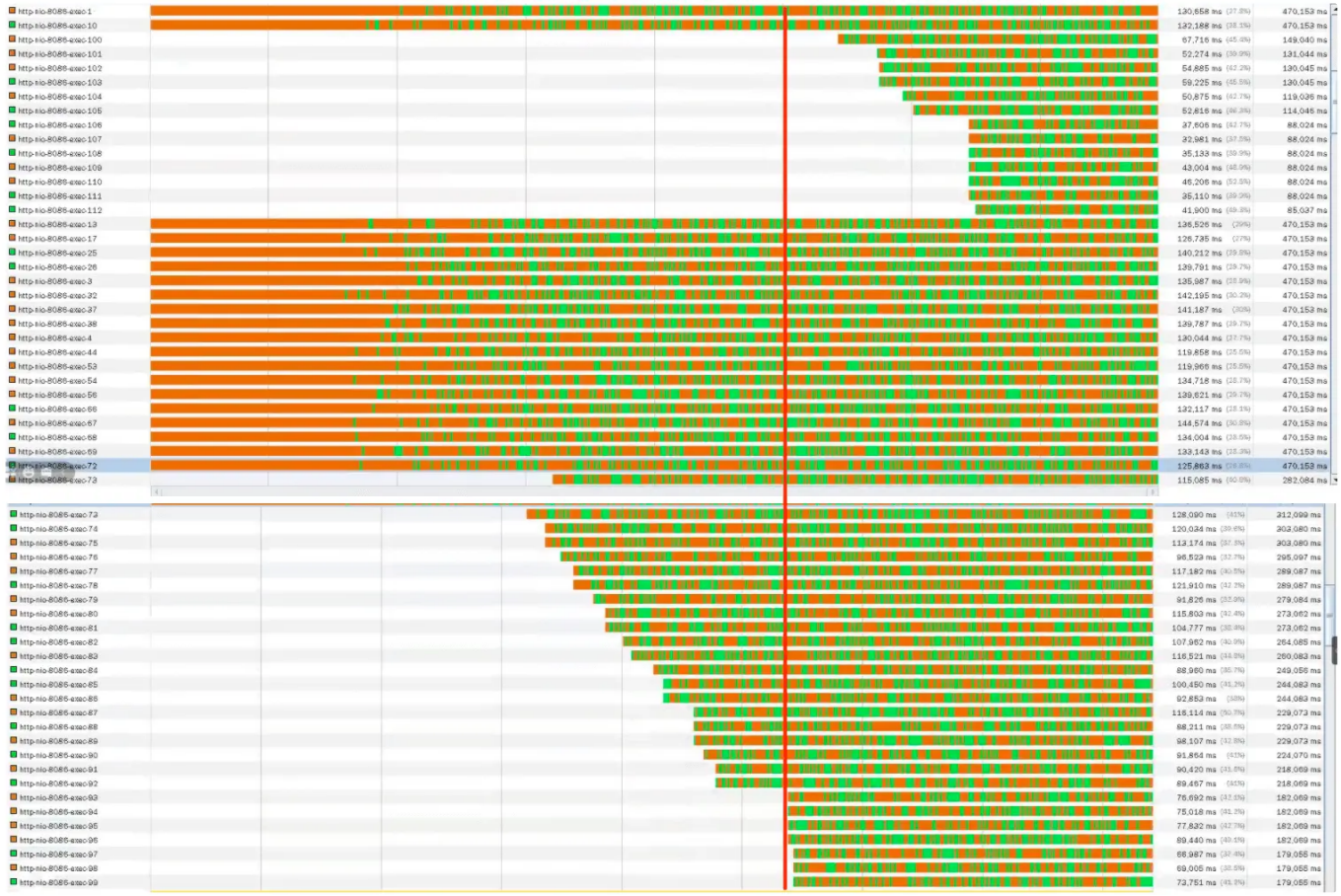
```
server:
  port: 8086
  tomcat:
    accept-count: 10000
    threads:
      max: 200
```

```
7       min-spare: 20
8     max-connections: 500
```

在没有压力之前，应用线程的状态是这样的：



压力起来之后，应用线程的状态是这样：



从线程的数量来看，线程数是在自适应增加的。对应压力中的 TPS 曲线和响应时间增加的地方，我们可以看到大概 41 个工作线程。随着压力的持续增加，TPS 还在增加，但是，响应时间慢慢变长了。从提供服务的角度来说，用户会感觉系统在逐渐变慢。

如果为了保证系统在生产上，用户的响应时间不想因为用户量的增加而变慢，这时候我们就可以考虑在这个服务中加上限流的手段了。

而对于我们这节课要确认的服务线程来说，我们想要支撑 800TPS 左右，其实只需要 41 个线程，所以，我们设置的 200 线程是用不到的。

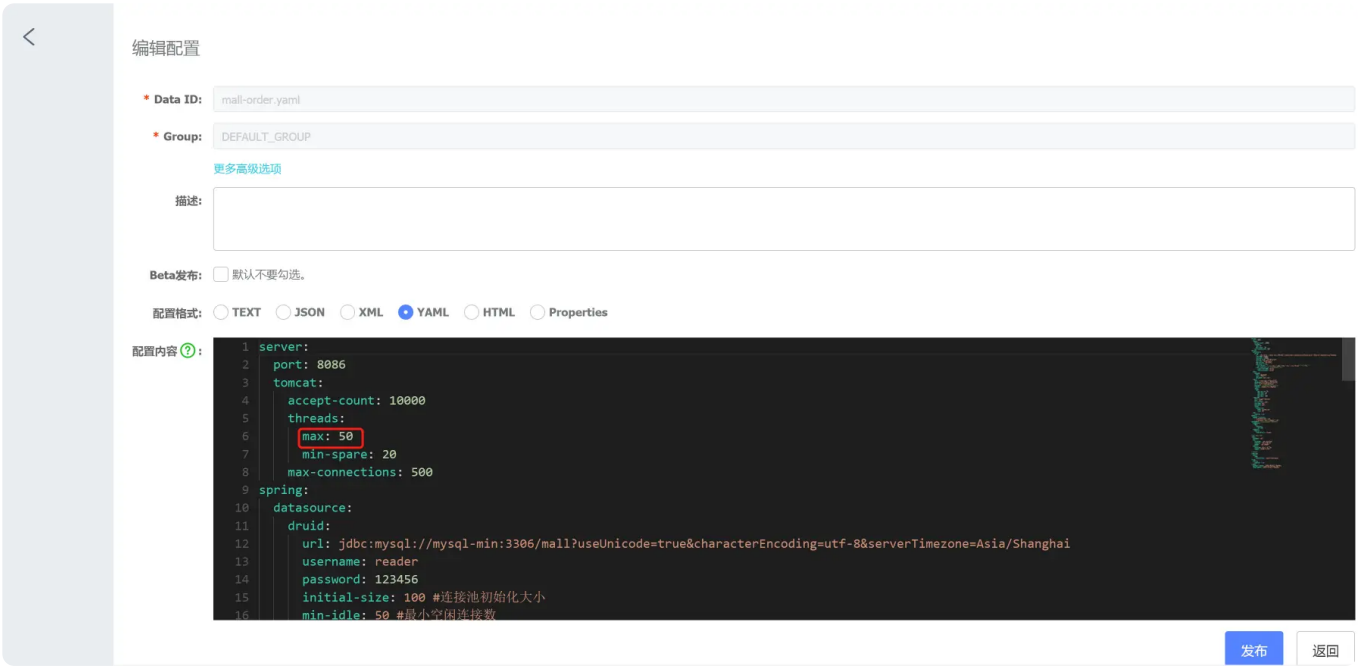
到这里，我们就确定了一个非常重要的性能参数——线程数，那我们应该把它配置为多少呢？

这时候，我们就得考虑一下，在这个服务中，我们想让 Order 服务支撑多少的容量？如果一个节点提供 800TPS 是可以接受的，并且对应的响应时间也都稳定，那我们就可以把线程数设置为稍高于 41 个线程，比如说 45 或 50 个线程。

你可能会想，200 远大于 41 个线程，把线程数直接设置成 200 不是更好吗？其实不是，如果我们要考虑峰值的流量，那么当流量大的时候，这个服务的响应时间会变长，直到超时退出，这给用户的感受显然是更糟糕的。因此，不建议做这样的配置。

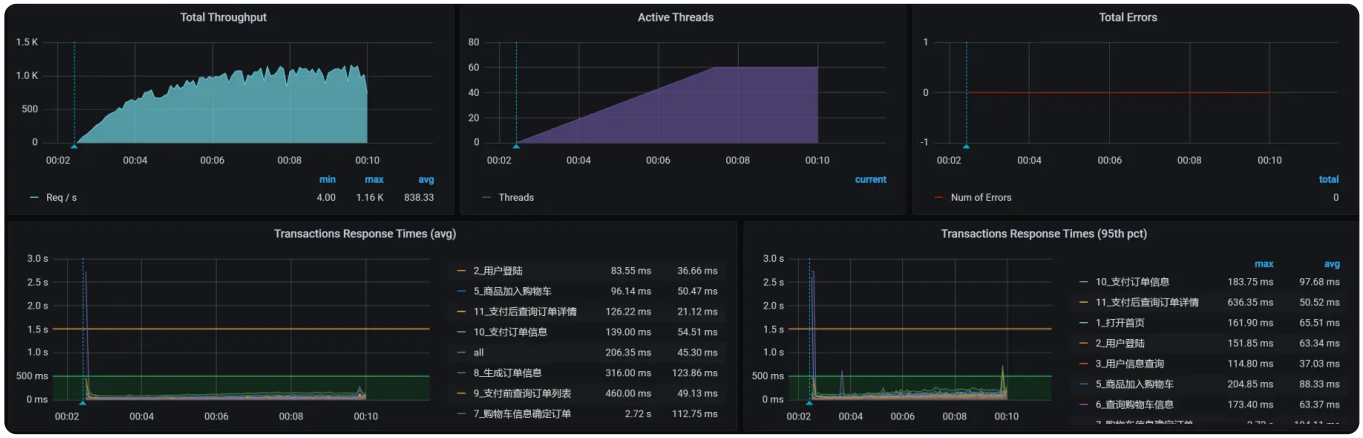
而更好的处理方式是，当这个服务不能提供稳定的响应时间，我们应该给用户一个友好的提示，这样不仅可以保证用户的访问质量，也能保证服务一直稳定。

现在，我在 Nacos 中把 max thread 改为 50，并发布配置：

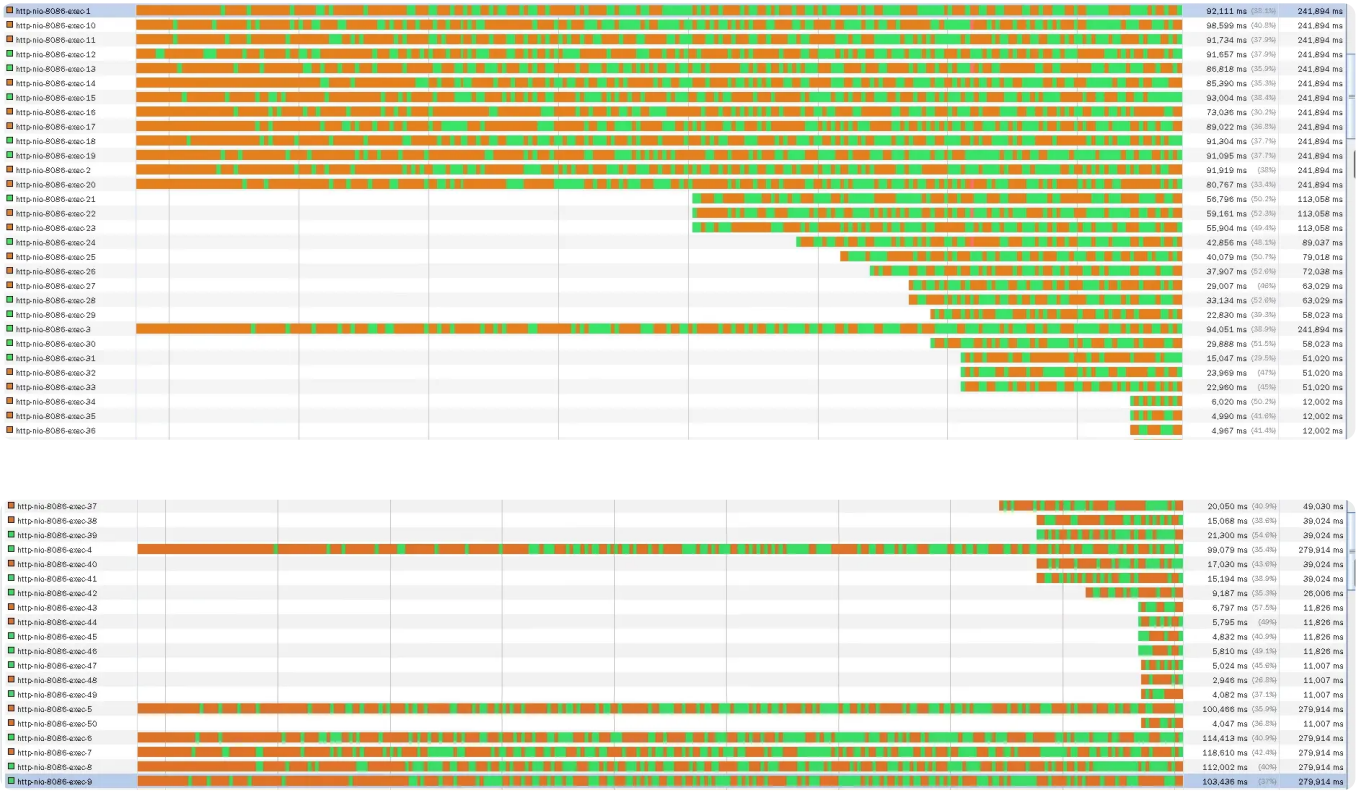


然后我们再重启 Order 服务。重启的时候你要注意，因为我们采用的是 Kubernetes 自动调度机制，所以我们要指定一下节点。如果不指定的话，重启之后的 POD 说不定会跑到其他的 worker 上去。我们还是要尽量保证两次测试处于同样的环境。

我们再执行一下场景看看：



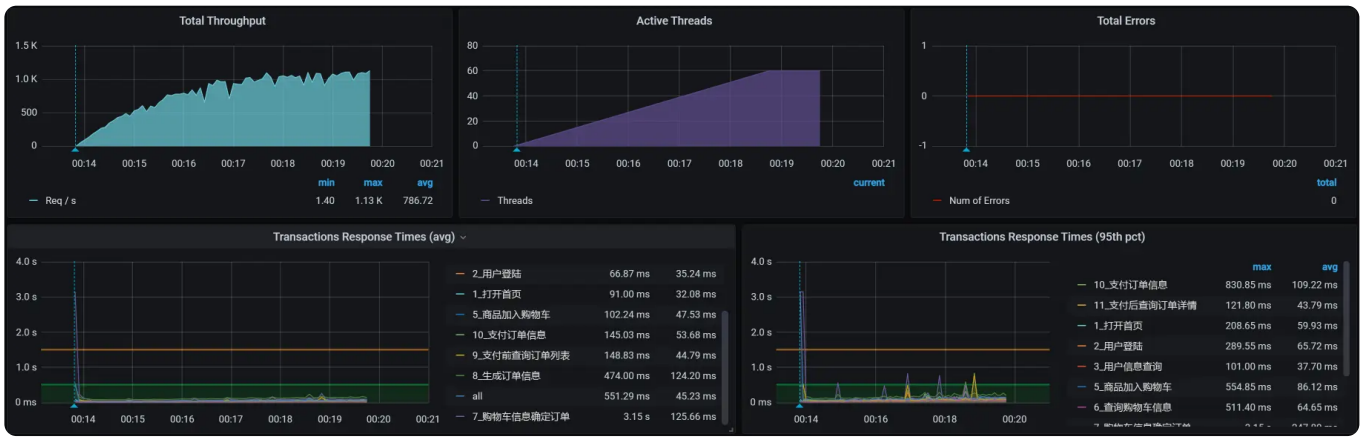
TPS 达到了 1000，我们再看一下线程数：



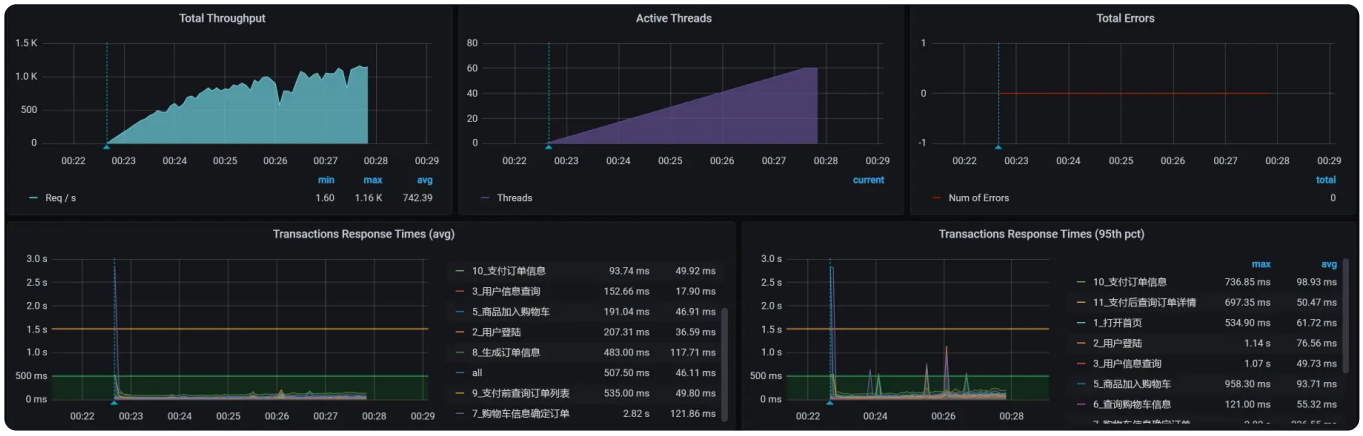
线程数正好是 50 个，也就是说 50 个线程就能支持到 1000TPS 了。

应用服务的超时和队列配置

而对于 Java 这样的应用服务，我们还需要考虑其他几个重要的性能配置参数，比如超时、队列等，这一点我们在前面的配置树中也有罗列。现在我们在保持 50 个线程的同时，再改一下队列长度。我们在上面看到的 accept-count 是 10000，为了让试验有效果，我们直接降为 1000，然后看看压力场景效果：



你看，还是能达到 1000TPS。那我们再把 accept-count 降下来一些，这次我们降狠一点，直接降为 10，希望达到因为队列不够长而产生报错的效果，来看下效果：



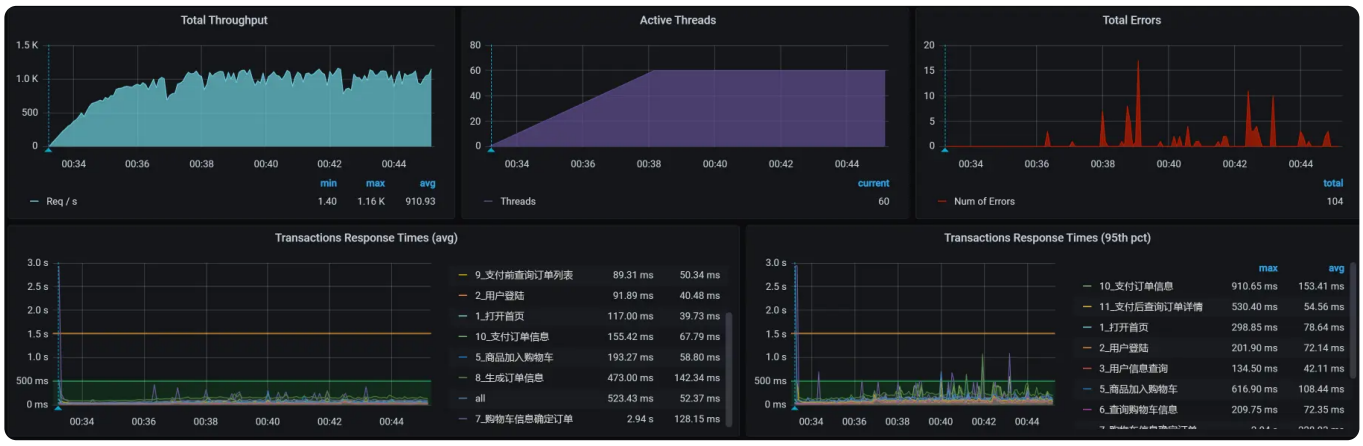
咦，怎么还没有报错？哦，是我大意了，没有设置超时。

那我们就增加一个参数 connection-timeout。在 Spring Boot 默认的 Tomcat 中，connection-timeout 是 60s。现在，我直接把它设置为 100ms，因为我们 Order 服务的响应时间有超过 100ms 的时候：

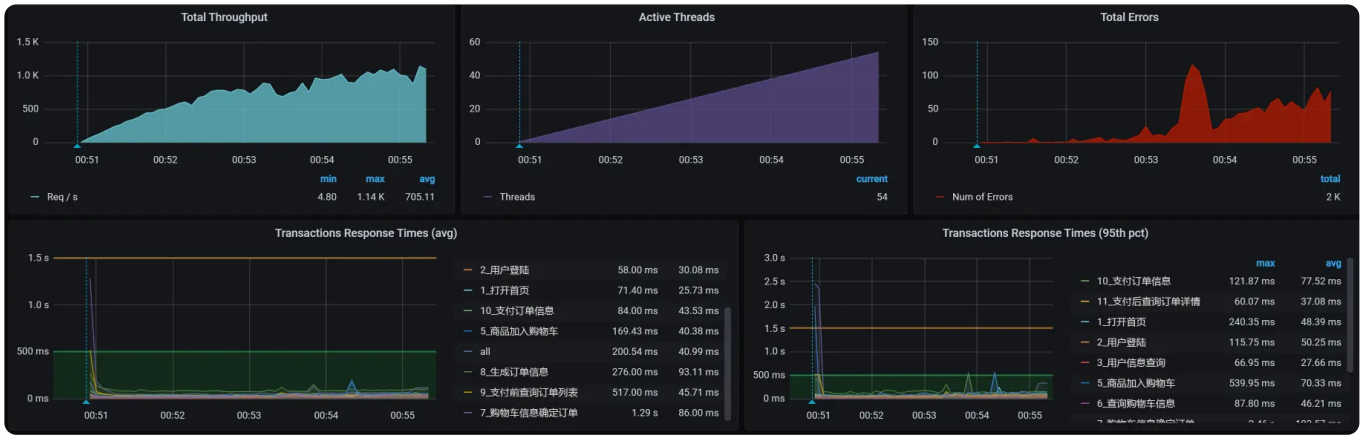
```yaml
server:
  port: 8086
  tomcat:
    accept-count: 10
    threads:
      max: 50
      min-spare: 20
    max-connections: 1000
    connection-timeout: 100
spring:
  datasource:
    druid:
      url: jdbc:mysql://mysql-min:3306/mall?useUnicode=true&characterEncoding=utf-8&serverTimezone=Asia/Shanghai
      username: reader
      password: 123456
      initial-size: 100 #连接池初始化大小
```

发布 返回

我们再次执行场景，看一下结果：



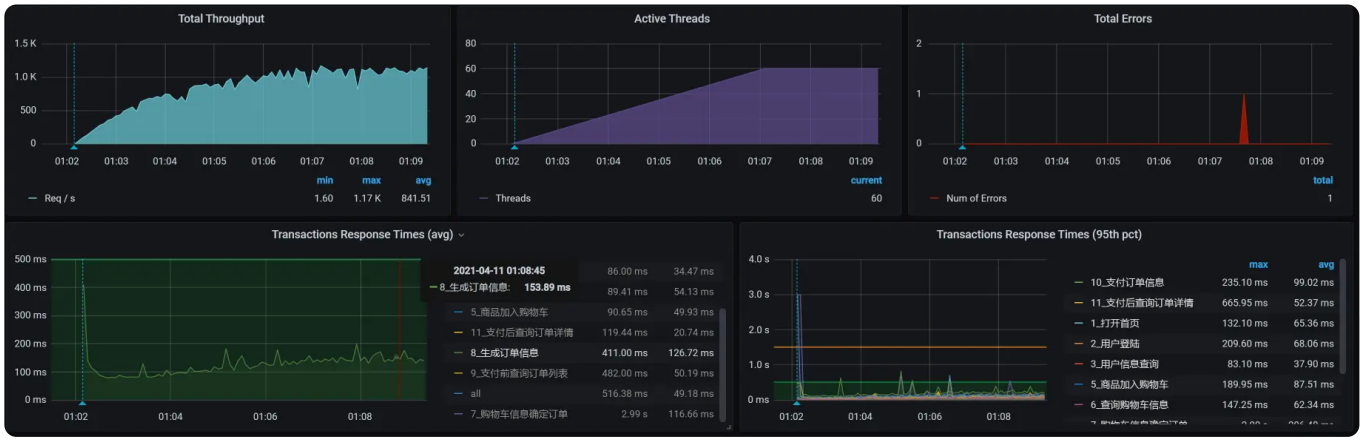
你看，报错了吧。这说明队列为 10、超时为 100 的设置过小了，无法保持每个请求都能正常返回。现在，我们把队列设置为 100，再来看一下：



看到没有，报错更多了，这符合我们的预期。因为队列长了，超时又短，队列中超时的请求自然会变多。并且在上面的曲线中，我们也可以看到，报错增加了不少。

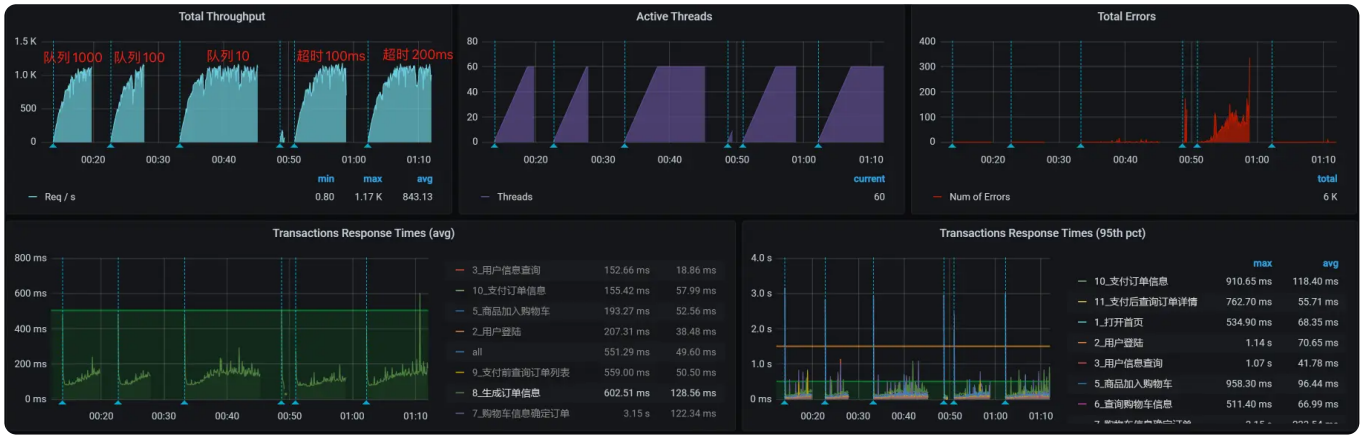
那怎么配置超时时长呢，我们要做的就是把超时增加，增加到大于响应时间中的最大值，只有这样才能不报错。

我们在上面的结果中看到，响应时间基本在 200ms 以下，那我们就把超时设置为 200ms，看一下结果：



你看，报错少了很多。这说明超时在性能调优中是一个很重要的参数，而它又和队列长度相关。

我们把前面的几个场景的结果都放到一个图中看一下：



通过这样一张图，我们就能清楚地看到线程池（线程数）大小 、超时、队列长度在不同设置下产生的效果比对。

因此，在这个应用中，我们可以设置的关键参数是：

参数	值
线程池	50
超时	200ms
队列	100

在这样的配置下，在加上限流、降级、熔断等手段，我们要保证的是，到这个服务的请求在 1000TPS 以内。

如果你想让这个系统在牺牲响应时间的前提下支撑更多的请求，就可以把上面的参数调大一些，具体调大到多少，就取决于你是想让系统支撑更多的请求，还是想让用户有更好的体验了。

总结

通过这节课，我给出了确定生产系统配置的思路。而做这件事情的前提是，我们对被测环境有明确的容量预期。在有了容量预期，并且对系统进行了调优之后，我们就可以通过这两个步骤把各个性能参数确定下来：

1. 发起压力；
2. 通过监控和场景执行数据，判断每个重要的性能参数的具体配置值。我强调一下，这一步需要我们非常细心，试验也要做很多遍。

由于性能相关参数有很多，这就需要我们结合性能配置树中罗列出的每个性能配置，一一确定。你可能会觉得这是一个非常费时费力的活。其实在一个项目中，这个步骤只需要全面地做一次，在后面的版本变更中，我们只需要根据性能分析的结果做相应的更新就可以了。并且在大部分项目中，这种更新不会出现大面积的参数变动情况。

课后作业

最后，请你思考一下：

1. 为什么性能项目中要做性能参数配置的确定？
2. 如何确定数据库及其他技术组件的性能参数呢？

记得在留言区和我讨论、交流你的想法，每一次思考都会让你更进一步。

如果你读完这篇文章有所收获，也欢迎你分享给你的朋友，共同学习进步。我们下这节课再见！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

赞 0 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 我们这个课程的系统是怎么搭建起来的？

下一篇 31 | 怎么写出有价值的性能报告？

精选留言 (3)

写留言

张东炫
2021-06-04

如果我们测试环境有 300C 资源，使用率也为 30%，要是我还想保证 0.1 秒的平均响应时间，那么 TPS 就应该是 300。这是最简单的等比方式了。
文中这里写的 TPS 是不是3000？

作者回复: 我算了一下。确实是手抖了一下少打了个零。可以联系我一下，我发红包感谢指正。

1

道长
2021-06-01

1、可以给项目组提供容量与资源的一个参考，大多时候做性能都是针对未来几年做的评估。避免浪费资源

作者回复: 理解正确。

Technological life
2021-05-31

cpu资源那列数据展示1000c，这个c是什么单位？

展开

作者回复: 就是CPU个数。